SHARP APL *for MVS*

Guide for APL Programmers

Version 21

SOLITON ASSOCIATES

SHARP APL is a registered trademark of Soliton Associates Limited.
NSVP is a trademark of Soliton Associates Limited.
All other products referred to in this document are trademarks or registered trademarks of their respective companies.

© 1994 by Soliton Associates Limited — Printed in Canada
44 Victoria Street, Suite 2100, Toronto, Ontario, Canada M5C 1Y2

All rights reserved. No part of this work may be reproduced in any form or by any means without prior written permission of the copyright owner.

Soliton Associates Limited shall not be liable for errors contained herein or for incidental consequential damages in connection with the use of this material. The information contained within this document is subject to change without notice.

Publication code: MM-000-1501

Guide for APL Programmers

Introduction

SHARP APL Version 21 for MVS ("V21") provides a number of enhancements to the SHARP APL product, resulting in improved performance, increased availability, and enhanced operability of the SHARP APL system. These changes include modifications and additions to the base APL system and to the file system, as well as changes to the SHARP APL utilities and other related software products.

This guide documents the changes that are of interest to APL programmers. These include

- association of assembly language routines with APL workspace objects via *intrinsic functions*
- a new language primitive *less* (~)
- a new file system verb, `⎕copy`
- 8-bit support for asynchronous terminals via changes to `⎕arbin` and `6I14`
- improved S-task communications via changes to `2 ⎕ws 3` and `⎕runs`
- miscellaneous documentation changes to the *SHARP APL Reference Manual.*

Intrinsic Functions

An intrinsic function is a formally defined routine whose definition may be part of the APL system, but which does not have any permanently assigned symbol or name, such as `⍴` or `⎕read`. This type of routine is accessed only by association with a symbol or name in the workspace. The association between an APL symbol or name and an intrinsic function is called an *intrinsic function name bind* (IFN bind), which is a physical object, resident within the APL workspace, created via the intrinsic function `system.bind` and displayed

using the intrinsic function *system.bindrep*. Like most other workspace objects, IFN binds can be saved along with the workspace and they can be copied and erased; however, they cannot exist in packages.

Intrinsic functions are most commonly used to improve performance of SHARP APL applications. For example, APL verbs that may be causing performance problems can be re-implemented in IBM/370 assembly language and accessed using the intrinsic function facility.

A number of useful intrinsic functions are distributed with V21. These are documented fully in the workspace *1 if*. Please contact your Soliton Associates representative regarding application and implementation of custom intrinsic functions.

Less (~)

Dyadic tilde (~)is defined as the new language primitive *less*. Its result is the array consisting of the major cells of α that are not also major cells of ω. Formally,

$$\alpha \sim \omega \quad \Leftrightarrow \quad (\sim(MC\ \alpha) \in MC\ \omega) \neq \alpha$$

where *MC* is $,<\overset{\cdot\cdot}{\circ}{}^{-}1\ \omega$

For example:

```
      0 1 2 3 4 5 ~ 7 2 4
0 1 3 5

      a ← 3 4 ⍴ 'abcdefghijkl'
      b ← 2 4 ⍴ 'mnopabcd'
      a ~ b
efgh
ijkl
```

As can be seen from the formal definition, less is sensitive to the value of ⎕*ct*, the comparison tolerance.

The result of *a~a~b* is called the intersection of *a* and *b*. For example:

```
      a ← 0 1 2 3 4 5
      b ← 7 4 2
      a~a~b
2 4
```

File Copy Verb ⎕*copy*

The new file verb ⎕*copy* copies a tied file even when that file is in use. The syntax for this verb is:

'(libno) newname' ⎕copy tieno(,passno)(,lpassno)

where *newname* is the name of the new file to be created, *libno* is the optional library number of the new file, *tieno* is the file tie number of the source file, *passno* is the optional passnumber of the source file, and *lpassno* is the optional passnumber for the library to which the file will be copied.

The contents of *newname* will match those of the source file as they were at the time □*copy* terminated. The new file is stored contiguously, if possible, thus reducing storage overhead.

The copy operation is functionally transparent to users of the source file; however, response time may be degraded due to contention for the file.

Library permission: 131072
File permission: 64

Example:

```
'test' □stie 99

'testfile' □copy 99

'1234567 newtest' □copy 99 100 3691
```

□*arbin* with Asynchronous Terminals

The system verb □*arbin* now supports 8-bit arbitrary input for asynchronously connected terminal devices. This has been achieved through extensions to the system verb 6I, as described below.

- 6I14 0 sets the default 7-bit □*arbin* input for ASCII-type devices
- 6I14 1 sets 8-bit □*arbin* input for ASCII-type devices.

For more information on 6I verbs, see the *SHARP APL Reference Manual.*

Improved S-task Communications

The inquiry verbs □*runs* and 2 □*ws* have been extended to better support S-task communications via NSVP.

- The third column of □*runs* now displays the remote user number of the task's initiator. Previously, for tasks controlled via NSVP this value was always reported as 13.
- The first element of 2 □*ws* 3 now holds the neighborhood ID through which the session is being managed. If the share is local (i.e., not via NSVP), this value is ¯1.

For more information on S-task communications via NSVP, see the *NSVP User's Guide.* For general information on S-tasks, see the *SHARP APL Reference Manual.*

Documentation Changes

The following items are corrections or additions to the *SHARP APL Reference Manual*, 2nd edition. Where appropriate, page references are provided.

Under Requires a Known Inverse

On page 7–16, the list of primitive monads and their inverses that can be used as a right argument to *under* (⍤) should read as follows:

Monad:	⋆	⍟	>	<	−	÷	⍲	⍱	~	⊢
Inverse:	⍟	⋆	<	>	−	÷	⍲	⍱	~	⊢

`(2 ⎕ws 3)[4]`

On page 16–87, the system verb `(2 ⎕ws3)[4]` is incorrectly documented as being unused. The correct meaning of this element is as follows:

`(2 ⎕ws 3)[4]` `1` if hot continue was loaded at sign-on; remains `1` until sign-off.

`0` if no hot continue was loaded.

`6I12`

Page 16–90 incorrectly documents the availability of `6I12 1` and `6I12 3`. These are in fact not supported. The only two variations of `6I12` that are supported are described below.

`6I12 ¯1` Same effect as `)keyb lock`.

`6I12 0` Same effect as `)keyb nomsg`.

Reload User Profile: `)proj`

Documentation for the system command `)proj` was inadvertently omitted from the *SHARP APL Reference Manual*. This command causes an account's `Profile` workspace to be loaded. This workspace is normally loaded when the task signs on, but may be skipped when a "hot continue" workspace is loaded instead.

Performing `)proj` is equivalent to performing `)load Profile`.